Lightweight Verification of Markov Decision Processes with Rewards

Axel Legay, Sean Sedwards and Louis-Marie Traonouez

Inria Rennes – Bretagne Atlantique

Abstract Markov decision processes are useful models of concurrency optimisation problems, but are often intractable for exhaustive verification methods. Recent work has introduced lightweight approximative techniques that sample directly from scheduler space, bringing the prospect of scalable alternatives to standard numerical algorithms. The focus so far has been on optimising the probability of a property, but many problems require quantitative analysis of rewards. In this work we therefore present lightweight verification algorithms to optimise the rewards of Markov decision processes. We provide the statistical confidence bounds that this necessitates and demonstrate our approach on standard case studies.

1 Introduction

Markov decision processes (MDP) describe systems that interleave nondeterministic actions and probabilistic transitions. Such systems may be seen as comprising probabilistic subsystems whose transitions depend on the states of the other subsystems, while the order in which concurrently enabled transitions execute is nondeterministic. This order may radically affect the system's behaviour and can be optimised. By assigning numerical rewards or costs to execution traces, MDPs have proven useful in many real optimisation problems [27]. More recently, in the context of formal verification, logics have been extended to allow model checkers to consider rewards [19].

Fig. 1 shows a typical fragment of an MDP. Its execution semantics are as follows. In a given state (s_0), an action ($a_1, a_2, \dots$) is chosen nondeterministically to select a distribution of probabilistic transitions ($p_1, p_2, \dots$ or p_3, p_4, etc). A probabilistic choice is then made to select the next state ($s_1, s_2, s_3, s_4, \dots$). In the classic case, rewards are assigned to actions [2,26,1]. In the context of model checking, rewards are often assigned to states or transitions between states [19]. In both cases the rewards are summed over the length of a trace and averaged over all traces, giving the expected reward. The mechanism of accumulating rewards is unimportant to our algorithms and we need only assume that a total reward is assigned to a finite trace.

In this work we focus on MDPs in the context of statistical model checking (SMC). Model checking is an automatic technique to verify that a system satisfies a property specified in temporal logic [7]. Probabilistic (numerical) model

checking quantifies the probability that a probabilistic system will satisfy a property [9]. Statistical model checking describes a collection of Monte Carlo methods that approximate the results of probabilistic model checking.

Numerical model checking algorithms to solve purely probabilistic systems scale exponentially with the number of interacting variables in the model [3]. Numerical algorithms to find extremal schedulers in MDPs incur the additional cost of (effectively) considering all the possible ways that the nondeterminism might be resolved [19]. While memory-efficient ("lightweight") Monte Carlo techniques have been developed to address the probabilistic problem (i.e., SMC), until recently [23] it has not been possible to address the nondeterministic problem in this way. Building on the techniques introduced in [23], in this work we present algorithms to find schedulers that (approximately) maximise or minimise the expected reward of finite traces in Markov decision processes. We use the notion of "smart sampling" to increase the efficiency of the simple sampling strategies given in [23] and present statistical confidence bounds to find the maximum or minimum reward amongst multiple estimates. To demonstrate our approach, we have implemented the algorithms in our statistical model checking platform, PLASMA [15,5][1], and apply them to a number of case studies from the literature.

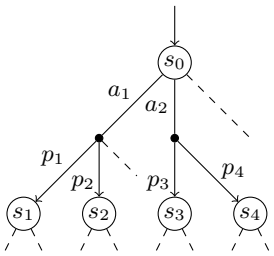

Figure 1. Fragment of a Markov decision process.

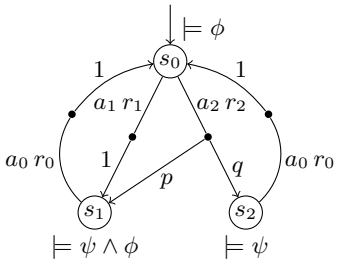

Figure 2. MDP with different rewards for general and memoryless schedulers.

1.1 Schedulers and the State Explosion Problem

Given an MDP with set of actions A, having a set of states S that induces a set of sequences of states $\Omega = S^+$, a history-dependent (general) scheduler is a function $\mathfrak{S} : \Omega \to A$. A memoryless scheduler is a function $\mathfrak{M} : S \to A$. Intuitively, at each state in the course of an execution, a general scheduler ($\mathfrak{S}$) chooses an action based on the sequence of previous states and a memoryless scheduler ($\mathfrak{M}$) chooses an action based only on the current state. History-dependent schedulers therefore include memoryless schedulers. We assume that rewards are defined by a function $R : \mathfrak{S} \times \Omega \to \mathbb{Q}$ that maps (*scheduler, trace*) pairs to total reward.

[1] projects.inria.fr/plasma-lab

In practice, total rewards are usually defined by the accumulation of rewards assigned to actions, states or transitions.

Fig. 2 illustrates a simple MDP for which memoryless and history-dependent schedulers can give different minimum rewards for logical property $\mathbf{X}(\psi \wedge \mathbf{X}\mathbf{G}^t\phi)$ and $t > 0$. The property makes use of the linear temporal operators *next* ($\mathbf{X}$) and *globally* ($\mathbf{G}$). Intuitively, the property states that on the next step ψ will be true and, on the step after that, ϕ will be remain true for $t+1$ time steps. In this example, rewards (r_0, r_1, r_2) are assigned to actions $(a_0, a_1, a_2$, respectively). In the initial state (s_0), both actions a_1 and a_2 can lead to traces that satisfy the property, but subsequent actions taken in state s_0 must be a_1. If $r_1 > r_2 > 0$, the minimum reward will be achieved by taking action a_2 in the initial state and a_1 whenever the execution visits s_0 thereafter. To satisfy the property, a memoryless scheduler would be forced to always take action a_1 in state s_0 and would therefore not achieve the minimum possible reward.

The principal challenge in finding optimal schedulers using numerical techniques is what has been described as the 'curse of dimensionality' [2] and the 'state explosion problem' [7]: the number of states of a system increases exponentially with respect to the number of interacting components and state variables. This phenomenon has led to the design of sampling algorithms that find 'near optimal' schedulers to optimise rewards in discounted MDPs [17] and motivates the present work.

The state explosion problem of model checking purely probabilistic systems has been well addressed by SMC [28]. SMC uses an executable model to approximate the probability that a system satisfies a specified property by the proportion of simulation traces that individually satisfy it. The state space of the system is not constructed explicitly – states are generated on the fly during simulation – hence SMC is "lightweight" and applicable to large, possibly infinite state, systems. As a consequence, SMC and may be easily and efficiently divided on parallel computing architectures.

SMC cannot be applied to MDPs without first resolving the nondeterminism. Since nondeterministic and probabilistic choices are interleaved in an MDP, memoryless schedulers are typically of the same order of complexity as the system as a whole and may be infinite. History dependent schedulers are exponentially bigger. With the exception of [23], previous attempts to apply SMC to MDPs have been "heavyweight", in that they store large data structures explicitly related to the states of the system. No previous approach has attempted to approximate the rewards based algorithms of numerical model checking.

1.2 Our Approach

Using a hash function, we implement general schedulers by mapping the concatenation of an integer seed and a trace to a pseudo-randomly chosen action. By randomly sampling seeds, we are thus able to randomly sample schedulers. To find optimal schedulers we use "smart sampling" to make efficient use of a finite simulation budget. Part of the budget is used to generate a candidate set of schedulers that has high probability of containing a near optimal scheduler.

The remaining budget is used to test and refine the candidate set, according to the specific confidence bounds we define for multiple estimates. Sub-optimal schedulers are removed from the candidate set and their budget is re-allocated to good ones. Hence, the performance of good schedulers is known with increasing confidence as the algorithm proceeds, until the best is known with sufficient confidence.

2 Related Work

There is considerable work on sampling algorithms to optimise rewards in discounted MDPs (see, e.g., [6] for a survey), however this notion of rewards is significantly different to the one commonly used in model checking [19]. We nevertheless describe the Kearns algorithm [17], since it is the basis of a recent application of SMC to MDPs [22].

The Kearns algorithm is the classic 'sparse sampling algorithm' for large, infinite horizon, discounted MDPs. It constructs a 'near optimal' scheduler piecewise, by approximating the best action from a current state, using a stochastic depth-first search. The algorithm can work with large, potentially infinite state MDPs because it explores a probabilistically bounded search space. This, however, is exponential in the discount. To find the action with the greatest expected reward in the current state, the algorithm recursively estimates the rewards of successive states, up to some maximum depth defined by the discount and desired error. Actions are enumerated while probabilistic choices are explored by sampling, with the number of samples set as a parameter. By iterating local exploration with probabilistic sampling, the discount guarantees that the algorithm eventually converges. The stopping criterion is when successive estimates differ by less than some error threshold.

There have been several recent attempts to apply SMC to MDPs [4,22,11,10,23]. None address the specific problem of model checking with rewards, although [22] makes use of the Kearns algorithm.

In [4,10] the authors present on-the-fly algorithms to remove 'spurious' nondeterminism from MDPs, so that standard SMC may be used. This approach is limited to the class of models whose nondeterminism does not affect the resulting probability of a property. The algorithms therefore do not attempt to address model checking problems related to finding optimal schedulers.

In [11] the authors present an algorithm to decide an hypothesis about the probability of an MDP property. Rewards are not considered. The algorithm generates candidate schedulers by counting state-action pairs in simulations, to iteratively improve a probabilistic scheduler. The candidates are assessed using sequential hypothesis testing and the process is repeated until an example is found or sufficient attempts have been made. If found, the example is correct, but state-actions are usually shared by many schedulers, so the improvement process may actually diverge from good schedulers.

In [22] the authors use an adaptation of the Kearns algorithm to find a memoryless scheduler that is near optimal with respect to a discounted reward

scheme. By storing information about visited states, the algorithm improves on the performance of the original Kearns algorithm, but is limited to memoryless schedulers of tractable size. The resulting scheduler induces a Markov chain whose properties may be verified with standard SMC. By considering only discounted rewards, however, [22] does not address the standard model checking problems of MDPs with rewards.

As a plausible variant of [22], it may seem reasonable to use the bounded properties of SMC to restrict the length of traces in the Kearns algorithm, rather than doing this implicitly with discounted rewards. The results would then be a correct approximation of numerical model checking algorithms for rewards. Unfortunately, since the Kearns algorithm relies on enumerating actions, this approach is effectively equivalent to enumerating schedulers and is not tractable.

With the exception of the original Kearns algorithm, the above approaches use data structures whose size scales with the state space of the MDP. We therefore consider these approaches to be "heavyweight". Very recently, the elements of lightweight verification of MDPs were introduced in [23]. By sampling directly from history-dependent scheduler space using only $\mathcal{O}(1)$ memory, [23] opens up the possibility of scalable verification of MDPs on parallel lightweight computational threads. The techniques of [23] form the basis of the present work.

3 Statistical Model Checking with Rewards

SMC algorithms work by constructing an automaton to decide whether a simulation trace $\omega \in \Omega$ satisfies property φ, denoted $\omega \models \varphi$. Since simulation traces are necessarily finite, property φ is typically expressed in a time bounded temporal logic (see, e.g., [16]). The automaton can then be used to estimate the probability of the property or to decide an hypothesis about the probability. In particular, the expected probability of φ is estimated by $\frac{1}{N}\sum_{i=1}^{N} \mathbf{1}(\omega_i \models \varphi)$, where $\omega_1, \ldots, \omega_N$ are N simulation traces selected uniformly at random from Ω and $\mathbf{1}(\cdot) \in \{0, 1\}$ is an indicator function corresponding to the output of the automaton: it returns 1 if the trace is accepted and 0 if it is not. To bound the estimation error, it is common to use the "Chernoff" bound of [25]. The user specifies an absolute error ε and a probability δ to define the bound $\mathrm{P}(\mid \hat{p} - p \mid \geq \varepsilon) \leq \delta$, where p and $\hat{p}$ are respectively the true probability and estimated probabilities. The bound is guaranteed if the number of simulations N satisfies the relation:

$$N \geq \left\lceil (\ln 2 - \ln \delta)/(2\varepsilon^2) \right\rceil . \tag{1}$$

The same notions can be used to estimate the expected reward. Given a function $R(\omega) \in [a, b]$, a, b finite, that assigns a total reward to simulation trace ω, the expected reward is estimated by $\frac{1}{N}\sum_{i=1}^{N} R(\omega_i)$. Since rewards may take values outside $[0, 1]$, we use Hoeffding's generalisation of (1) [12] to bound the errors. To guarantee $\mathrm{P}(\mid \hat{r} - r \mid \geq \varepsilon) \leq \delta$, where r and $\hat{r}$ are respectively the true and estimated values of expected reward, N is required to satisfy the relation:

$$N \geq \left\lceil \ln(2/\delta) \times (a - b)^2/(2\varepsilon^2) \right\rceil . \tag{2}$$

The values of a and b are usually unknown, hence (2) cannot be implemented explicitly. We see, however, that N depends on the ratio of the absolute error ε to the range of values $(a - b)$. The confidence of estimates of rewards may therefore be specified as a percentage. For simplicity, our algorithms use (1), assuming ε expresses a percentage as a fraction of 1.

The rewards properties commonly used in numerical model checking are based on an extension of the logic PCTL [19]. This extension defines *instantaneous* rewards (the average reward assigned to the k^{th} state of all traces, denoted $\mathbf{I}^k$), *cumulative* rewards (the average total reward accumulated up to the k^{th} state of all traces, denoted $\mathbf{C}^k$) and *reachability* rewards (the average accumulated reward of traces that eventually satisfy property φ, denoted $\mathbf{F}\varphi$). Instantaneous and cumulative rewards are based on finite traces and can be immediately approximated by sampling, using (1) to bound the errors. Reachability rewards are based on unbounded $\mathbf{F}$ (the *finally* or *eventually* operator) and require additional consideration.

By the definition of reachability rewards [19], traces that do not satisfy the property are assigned infinite reward. Hence, if $\mathrm{P}(\mathbf{F}\varphi) < 1$ there exists a path that does not satisfy φ and the average reward is infinite. If $\mathrm{P}(\mathbf{F}\varphi) = 1$, all considered paths are finite and the average reward is also finite. Using sampling, it is not possible to say with certainty whether $\mathrm{P}(\mathbf{F}\varphi) = 1$, even if every observed trace of finitely many satisfies φ. Hence, the random variable from which samples are drawn could include the value infinity, giving it infinite variance. Error bounds that rely on finite variance, such as (1), (2) and the standard confidence interval, therefore cannot be applied directly.

Our solution is to implement $\mathbf{F}\varphi$ as $\mathbf{F}^k\varphi$, i.e., bounded reachability, with bound k set much longer than it is supposed necessary to satisfy φ. If all observed traces satisfy φ, the estimated average reward is within the limits defined by (1), given the hypothesis $\mathrm{P}(\mathbf{F}\varphi) = 1$ is true. If no counterexamples to $\mathbf{F}^k\varphi$ are observed, the confidence of this hypothesis increases with increasing numbers of simulations and may be quantified according to standard hypothesis tests. In practice, the simulations required to satisfy (1) typically lead to much greater confidence with respect to the hypothesis $\mathrm{P}(\mathbf{F}\varphi) = 1$. If a counterexample is observed, the hypothesis may nevertheless be accepted according to the specified confidence. In this case the user may either conclude that the average reward is infinite, accept the calculated average reward as a lower bound or increase k and try again.

To accommodate instantaneous and cumulative rewards bounded by k, we assume that all simulation traces reach the k^{th} state. This may be achieved by adding self loops to halting states, suitably augmented with rewards, as is standard practice in numerical model checking.

4 Lightweight Verification of MDPs

To avoid storing schedulers as explicit mappings, we construct schedulers on the fly using uniform pseudo-random number generators (PRNG) that are initialised

by a *seed* and iterated to generate the next pseudo-random value. In general, such PRNGs aim to ensure that arbitrary subsets of sequences of iterates are uniformly distributed and that consecutive iterates are statistically independent. PRNGs are commonly used to implement the uniform probabilistic scheduler, which chooses actions uniformly at random and thus explores all possible combinations of nondeterministic choices. Executing such an implementation twice with the same seed will produce identical traces. Executing the implementation with a different seed will produce an unrelated set of nondeterministic and probabilistic choices. It is therefore not possible to estimate the probability of a property under a specific scheduler from either $\mathfrak{M}$ or $\mathfrak{S}$.

Our solution is to use independent PRNGs to resolve nondeterministic and probabilistic choices. It is then possible to generate multiple probabilistic simulation traces per scheduler by fixing the seed of the PRNG for nondeterministic choices, while choosing random seeds for a separate PRNG for probabilistic choices. In a naive implementation, the sequence of iterates from the PRNG used for nondeterministic choices will be the same for all instantiations of the PRNG used for probabilistic choices, hence the i^{th} iterate of the PRNG for nondeterministic choices will always be the same, regardless of the state arrived at by the previous probabilistic choices. The i^{th} chosen action will be neither state nor trace dependent. To span the full range of general schedulers we therefore construct a per-step PRNG seed that is a *hash* of the integer identifying a specific scheduler concatenated with an integer representing the sequence of states up to the present.

4.1 General Schedulers Using Hash Functions

We assume that a state of an MDP is an assignment of values to a vector of system variables $v_i, i \in \{1, \ldots, n\}$. Each v_i is represented by a number of bits b_i, typically corresponding to a primitive data type (*int*, *float*, *double*, etc). The state can thus be represented by the concatenation of the bits of the system variables, such that a sequence of states may be represented by the concatenation of the bits of all the states. Without loss of generality, we interpret such a sequence of states as an integer of $\sum_{i=1}^{n} b_i$ bits, denoted $\overline{s}$, and refer to this in general as the *trace vector*. A scheduler is denoted by an integer σ, which is concatenated to $\overline{s}$ (denoted $\sigma : \overline{s}$) to uniquely identify a trace and a scheduler. Our approach is to generate a hash code $h = \mathcal{H}(\sigma : \overline{s})$ and to use h as the seed of a PRNG that resolves the next nondeterministic choice.

The hash function $\mathcal{H}$ thus maps $\sigma : \overline{s}$ to a seed that is deterministically dependent on the trace and the scheduler. The PRNG maps the seed to a value that is uniformly distributed but nevertheless deterministically dependent on the trace and the scheduler. In this way we can approximate the scheduler functions $\mathfrak{S}$ and $\mathfrak{M}$ described in Section 1.1. Importantly, the technique only relies on the standard properties of hash functions and PRNGs. Algorithm 1 is the basic simulation function used by our algorithms.

Algorithm 1: Simulate

Input:
$\mathcal{M}$: an MDP with initial state s_0
φ: a bounded temporal logic property
σ: an integer identifying a scheduler

Output:
ω: a simulation trace

Let $\mathcal{U}_{\text{prob}}, \mathcal{U}_{\text{nondet}}$ be uniform PRNGs with respective samples $r_{\text{pr}}, r_{\text{nd}}$
Let $\mathcal{H}$ be a hash function
Let s denote a state, initialised $s \leftarrow s_0$
Let ω denote a trace, initialised $\omega \leftarrow s$
Let $\overline{s}$ be the trace vector, initially empty
Select seed of $\mathcal{U}_{\text{prob}}$ randomly
while $\omega \models \varphi$ *is not decided* **do**
    $\overline{s} \leftarrow \overline{s} : s$
    Set seed of $\mathcal{U}_{\text{nondet}}$ to $\mathcal{H}(\sigma : \overline{s})$
    Iterate $\mathcal{U}_{\text{nondet}}$ to generate r_{nd} and use to resolve nondeterministic choice
    Iterate $\mathcal{U}_{\text{prob}}$ to generate r_{pr} and use to resolve probabilistic choice
    Set s to the next state
    $\omega \leftarrow \omega : s$

4.2 An Efficient Iterative Hash Function

To implement our approach, we use an efficient hash function that constructs seeds incrementally. The function is based on modular division [18, Ch. 6], such that $h = (\sigma : \overline{s}) \bmod m$, where m is a suitably large prime.

Since $\overline{s}$ is a concatenation of states, it is usually very much larger than the maximum size of integers supported as primitive data types. Hence, to generate h we use Horner's method [13][18, Ch. 4]: we set $h_0 = \sigma$ and find $h \equiv h_n$ (n as in Section 4.1) by iterating the recurrence relation

$$h_i = (h_{i-1} 2^{b_i} + v_i) \bmod m. \tag{3}$$

The size of m defines the maximum number of different hash codes. The precise value of m controls how the hash codes are distributed. To avoid collisions, a simple heuristic is that m should be a large prime not close to a power of 2 [8, Ch. 11]. Practically, it is an advantage to perform calculations using primitive data types that are native to the computational platform, so the sum in (3) should always be less than or equal to the maximum permissible value. To achieve this, given $x, y, m \in \mathbb{N}$, we note the following congruences:

$$(x + y) \bmod m \equiv (x \bmod m + y \bmod m) \bmod m \tag{4}$$

$$(xy) \bmod m \equiv ((x \bmod m)(y \bmod m)) \bmod m \tag{5}$$

The addition in (3) can thus be re-written in the form of (4), such that each term has a maximum value of $m - 1$:

$$h_i = ((h_{i-1} 2^{b_i}) \bmod m + (v_i) \bmod m) \bmod m \tag{6}$$

To prevent overflow, m must be no greater than half the maximum possible integer. Re-writing the first term of (6) in the form of (5), we see that before taking the modulus it will have a maximum value of $(m-1)^2$, which will exceed the maximum possible integer. To avoid this, we take advantage of the fact that h_{i-1} is multiplied by a power of 2 and that m has been chosen to prevent overflow with addition. We thus apply the following recurrence relation:

$$(h_{i-1}2^j) \bmod m = (h_{i-1}2^{j-1}) \bmod m + (h_{i-1}2^{j-1}) \bmod m \tag{7}$$

Equation (7) allows our hash function to be implemented using efficient native arithmetic. Moreover, we infer from (3) that to find the hash code corresponding to the current state in a trace, we need only know the current state and the hash code from the previous step. When considering memoryless schedulers we need only know the current state.

4.3 Estimating Multiple Schedulers

Our algorithms require that we find the optimum estimates of a number of schedulers. We sample M schedulers and generate M corresponding estimates $\{\hat{r}_1, \ldots, \hat{r}_M\}$, taking the maximum ($\hat{r}_{\max}$) or minimum ($\hat{r}_{\min}$), as required. To overcome the cumulative probability of error with the standard Chernoff bound (1) [23], we specify that *all* estimates $\hat{r}_i$ must be within ε of their respective true values r_i, ensuring that any $\hat{r}_{\min}, \hat{r}_{\max} \in \{\hat{r}_1, \ldots, \hat{r}_M\}$ are within ε of their true value. Given that estimates $\hat{r}_i$ are statistically independent, the probability that all estimates are less than their upper bound is expressed by $\mathrm{P}(\bigwedge_{i=1}^{M} \hat{r}_i - r_i \leq \varepsilon) \geq (1 - e^{-2N\varepsilon^2})^M$. Hence, $\mathrm{P}(\bigvee_{i=1}^{M} \hat{r}_i - r_i \geq \varepsilon) \leq 1 - (1 - e^{-2N\varepsilon^2})^M$, giving $N = \left\lceil -\ln\left(1 - \sqrt[M]{1-\delta}\right)/(2\varepsilon^2) \right\rceil$ for parameters M, ε and δ. This ensures that $\mathrm{P}(r_{\min} - \hat{r}_{\min} \geq \varepsilon) \leq \delta$ and $\mathrm{P}(\hat{r}_{\max} - r_{\max} \geq \varepsilon) \leq \delta$. To ensure the more usual stronger conditions that $\mathrm{P}(\mid r_{\max} - \hat{r}_{\max} \mid \geq \varepsilon) \leq \delta$ and $\mathrm{P}(\mid r_{\min} - \hat{r}_{\min} \mid \geq \varepsilon) \leq \delta$, we have

$$N = \left\lceil \left(\ln 2 - \ln\left(1 - \sqrt[M]{1-\delta}\right)\right)/(2\varepsilon^2) \right\rceil. \tag{8}$$

Note that when $M = 1$, (8) degenerates to (1). N scales logarithmically with M, making it tractable to consider many schedulers. The use of smart sampling, however, makes it unnecessary to test all candidate schedulers with maximum confidence.

5 Smart Sampling

The simple sampling strategies used in [23] have the disadvantage that they allocate equal simulation budget to all schedulers, regardless of their merit. The idea of smart sampling is to maximise the probability of finding an optimal scheduler with a finite simulation budget and not waste budget estimating schedulers that are not optimal.

In general, the problem of finding optimal schedulers using sampling has two independent components: the rarity of near optimal schedulers (denoted p_g) and the rarity of the property under near optimal schedulers (denoted $p_{\overline{g}}$). A near optimal scheduler is one whose reward or probability (depending on the context) is within some ε of the optimal value. If we select M schedulers uniformly at random and verify each with N simulations, the expected number of traces that satisfy the property using a near optimal scheduler is thus $Mp_gNp_{\overline{g}}$. The probability of seeing a trace that satisfies the property using a near optimal scheduler is the cumulative probability

$$(1 - (1 - p_g)^M)(1 - (1 - p_{\overline{g}})^N). \tag{9}$$

To maximise the chance of seeing a good scheduler with a simulation budget of $N_{\max} = NM$, N and M should be chosen to maximise (9). Then, following a sampling experiment using these values, any scheduler that produces at least one trace that satisfies φ becomes a candidate for further investigation. Since the values of of p_g and $p_{\overline{g}}$ are unknown a priori, it is necessary to perform an initial uninformed sampling experiment to estimate them, setting $N = M = \lceil\sqrt{N_{\max}}\rceil$. The results can be used to numerically optimise (9), however an effective heuristic is to set $N = \lceil 1/\hat{p}_{\overline{g}}\rceil$, where $\hat{p}_{\overline{g}}$ is the maximum observed estimate (or minimum non-zero estimate in the case of finding minimising schedulers).

The best scheduler is found by iteratively refining the candidate set. At each step, the per-step simulation budget ($N_{\max}$) is divided between the remaining candidates, simulations are performed and the average reward for each scheduler is estimated. Schedulers whose estimates fall into the "worst" quantile (lower or upper half, depending on context) are discarded. Refinement continues until estimates are known with specified confidence, according to (8). With a per-iteration budget satisfying (1), the algorithm is guaranteed to terminate with a valid estimate.

Finding schedulers that optimise the rewards defined in [19] is simpler than finding schedulers that optimise the probability of a property. This is because the effective probability of rewards properties is always one. In the case of instantaneous and cumulative rewards, traces are not filtered with respect to a property, so all traces are accepted. In the case of reachability rewards, either all traces satisfy the property or the reward is assumed to be infinite. Hence, the case of probabilities less than one does not have to be quantified, just detected. The consequence of this, according to (9), is that the simulation budget to generate the initial candidate set can be allocated entirely to schedulers, i.e., $N = 1$ and $M = N_{\max}$.

5.1 Smart Reward Estimation Algorithm

Algorithm 2 finds schedulers that maximise rewards. The algorithm to minimise rewards follows intuitively: replace instances of 'max' with 'min' in lines 16, 17, 21 and the Output line, and replace line 20 with $S \leftarrow \{\sigma \in S \mid \sigma = Q'(n) \wedge n \in \{1, \ldots, \lceil |S|/2 \rceil\}\}$.

Algorithm 2: Reward Estimation

Input:

$\mathcal{M}$: an MDP

$\rho \in \{\mathbf{I}^k\varphi, \mathbf{C}^k\varphi, \mathbf{F}^k\varphi\}$: a reward property

$\mathcal{R}_\rho$: the reward function for ρ

$H_0, z(\alpha)$: hypothesis $P(\mathbf{F}^k\varphi) \geq p_0$ and normal quantile of order α

ε, δ: the reward estimation Chernoff bound

$N_{\max} > \ln(2/\delta)/(2\varepsilon^2)$: the per-iteration budget

Output:

$\hat{r}_{\overline{\max}} \approx r_{\max}$, where $r_{\overline{\max}} \approx r_{\max}$ and $\mathrm{P}(|r_{\overline{\max}} - \hat{r}_{\overline{\max}}| \geq \varepsilon) \leq \delta$

$N \leftarrow 1$, $M \leftarrow N_{\max}$

$S \leftarrow \{M \text{ seeds chosen uniformly at random}\}$

$\forall \sigma \in S, \forall j \in \{1, \ldots, N\} : \omega_j^\sigma \leftarrow \text{Simulate}(\mathcal{M}, \varphi, \sigma)$

$Q \leftarrow \{(\sigma, q) \mid \sigma \in S \wedge \mathbb{Q} \ni q = \sum_{j=1}^{N} \mathcal{R}_\rho(\sigma, \omega_j^\sigma)/N\}$

$\forall \sigma \in S : \mathit{trues}(\sigma) \leftarrow 0$

$\mathit{samples} \leftarrow 0$, $\mathit{conf} \leftarrow 1$, $i \leftarrow 0$

while $\mathit{conf} > \delta \wedge S \neq \emptyset$ **do**

  $i \leftarrow i + 1$

  $M_i \leftarrow |S|$, $N_i \leftarrow 0$

  while $\mathit{conf} > \delta \wedge N_i < \lceil N_{\max}/M_i \rceil$ **do**

    $N_i \leftarrow N_i + 1$

    $\mathit{conf} \leftarrow 1 - (1 - e^{-2\epsilon^2 N_i})^{M_i}$

    $\forall \sigma \in S : \omega_{N_i}^\sigma \leftarrow \text{Simulate}(\mathcal{M}, \varphi, \sigma)$

    $\mathit{samples} \leftarrow \mathit{samples} + 1$

  $Q \leftarrow \{(\sigma, q) \mid \sigma \in S \wedge \mathbb{Q} \ni q = \sum_{j=1}^{N_i} \mathcal{R}_\rho(\sigma, \omega_j^\sigma)/N_i\}$

  $\sigma_{\max} \leftarrow \arg\max_{\sigma \in S} Q(\sigma)$

  $\hat{r}_{\overline{\max}} \leftarrow Q(\sigma_{\max})$

  $\forall \sigma \in S, j \in \{1, \ldots, N_i\} : \mathit{trues}(\sigma) = \mathit{trues}(\sigma) + \mathbf{1}(\omega_j^\sigma \models \varphi)$

  $Q' : \{1, \ldots, |S|\} \to S$ is an injective function s.t.
  $\forall (n, \sigma), (n', \sigma') \in Q' : n > n' \implies Q(\sigma) \geq Q(\sigma')$

  $S \leftarrow \{\sigma \in S \mid \sigma = Q'(n) \wedge n \in \{\lfloor |S|/2 \rfloor, \ldots, |S|\}\}$

$Z \leftarrow (\mathit{trues}(\sigma_{\max}) - \mathit{samples}\, p_0)/\sqrt{\mathit{samples}\, p_0\, (1 - p_0)}$

if $Z \leq z(\alpha)$ **then**

  H_0 is rejected

The reward property ρ may be of type instantaneous, cumulative or reachability, which are denoted $\mathbf{I}^k\varphi$, $\mathbf{C}^k\varphi$ and $\mathbf{F}^k\varphi$, respectively, to unify the operation of our algorithms. In the case of $\mathbf{I}^k\varphi$ and $\mathbf{C}^k\varphi$, k is user specified and φ is implicitly $\mathbf{G}^k\mathit{true}$. In the case of $\mathbf{F}^k\varphi$, φ is user specified and k is set as large as feasible to guarantee $\mathrm{P}(\mathbf{F}^k\varphi) = 1$. The reward function $\mathcal{R}_\rho : \mathbb{N} \times \Omega \to \mathbb{Q}$ maps the seed of a scheduler and a trace to a reward, given reward property ρ.

Typically, the per-iteration budget will be such that the required confidence is reached according to (8) before the candidate set is reduced to a single element. Lines 10 to 14 allow the algorithm to quit as soon as the minimum number of simulations is reached.

To avoid unnecessary complication, the mechanisms by which the test of hypothesis $\mathrm{P}(\mathbf{F}^k\varphi) \geq p_0$ would achieve confidence level α for arbitrary probability p_0 are not given. Instead, to illustrate the basic idea, in lines 21 to 23 we include a simple normal approximation hypothesis test, based on the number of samples used to estimate the rewards. This is typically more than adequate.

6 Case Studies

We implemented Algorithms 1 and 2 in our statistical model checking platform, PLASMA[1], and were thus able to take advantage of its distributed verification algorithm on the IGRIDA parallel computational grid[2]. All timings are based on 64 simulation cores. The following results demonstrate typical performance on a selection of standard case studies and reward properties taken from the numerical model checking literature. On these and many other examples not shown, we were able to achieve accurate results with a relatively modest per-iteration simulation budget of 10^5 simulations and a Chernoff bound of $\varepsilon = \delta = 0.01$. The models can be found illustrated in detail on the PRISM case studies website[3].

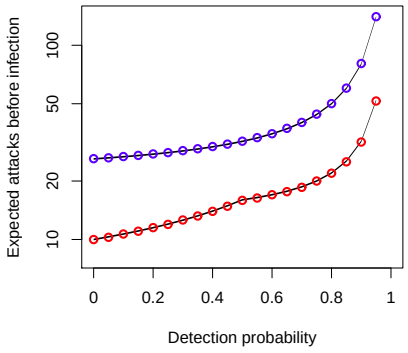

Figure 3. Network virus infection.

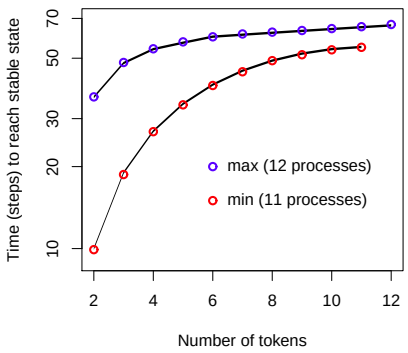

Figure 4. Self-stabilisation.

[2] igrida.gforge.inria.fr
[3] www.prismmodelchecker.org/casestudies

Network Virus Infection Our network virus infection case study is based on [21] and initially comprises the following sets of linked nodes: a set containing one node infected by a virus, a set with no infected nodes and a set of uninfected barrier nodes which divides the first two sets. A virus chooses which node to infect nondeterministically. A node detects a virus probabilistically and we vary this probability as a parameter for barrier nodes. Fig. 3 illustrates the results of estimating the maximum and minimum expected number of detected attacks before a particular node is infected. Each point required approximately 15 seconds of simulation time. In Figs. 3 and 4 the black lines show the true values, with their width indicating the specified $\pm 1\%$ error.

Self Stabilisation The self-stabilising protocol of [14] works asynchronously to ensure that a number of networked processes share a single 'privileged status' token fairly. The protocol is designed to reach this dynamical state even if initially there are several tokens in the ring. For models containing 11 and 12 processes, we estimated the expected number of steps to reach stability, given different initial numbers of tokens. Fig. 4 plots the maximum values for 12 processes and the minimum values for 11 processes. Individual estimates required between 1 and 3 minutes of simulation time.

Gossip Protocol The gossip protocol of [20] uses local connectivity to propagate information globally. Our algorithms correctly estimate the expected minimum and maximum number of rounds necessary for the network to become connected to be 1.486 and 4.5. The correct values are 1.5 and 4.5. The average simulation time per estimate was approximately 1 minute.

Choice Coordination To demonstrate the scalability of our approach, we consider instances of the choice coordination model of [24] with $BOUND = 100$. This value makes most of the models intractable to numerical model checking, however it is possible to infer the correct values of rewards from tractable instances. The property gives the expected minimum number of rounds necessary for a group of tourists to meet. The following table summarises the results:

Number of tourists	2	3	4	5	6	7	8	9	10
Minimum number of rounds to converge	4.0	5.0	7.0	8.0	10.0	11.0	12.0	13.0	14.0

All the estimates are exactly correct, while the average time to generate each result was just 8 seconds.

7 Prospects and Challenges

In this work we have focused on estimating the values of optimal rewards, but our techniques are immediately extensible to sequential hypothesis testing. One advantage of hypothesis testing is that schedulers which are found to satisfy the

hypothesis are true examples with the specified confidence. In the case of estimation, the specified confidence is with respect to the estimate, not the optimality. Quantifying the optimality of estimates is an ongoing challenge.

Our case studies demonstrate that our approach is effective and can be efficient, but for a given hash function and PRNG, the presented algorithms typically sample from only a subset of possible schedulers. This can be addressed by also sampling from hash functions and PRNGs, however it is easy to construct examples where near optimal schedulers are vanishingly rare. This makes them difficult to find by sampling alone and motivates the development of lightweight learning techniques that accelerate convergence and draw from a larger set of schedulers. To avoid storing per-state information, or creating similar heavyweight data structures, in future work we propose to construct schedulers piecewise, by modifying and combining hash functions and PRNGs in response to information gained from simulation.

Acknowledgement

This work was partially supported by the European Union Seventh Framework Programme under grant agreement no. 295261 (MEALS).

References

1. C. Baier and J.-P. Katoen. *Principles of model checking*. MIT Press, 2008.
2. R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.
3. A. Bianco and L. De Alfaro. Model checking of probabilistic and nondeterministic systems. In *Foundations of Software Technology and Theoretical Computer Science*, pages 499–513. Springer, 1995.
4. J. Bogdoll, L. M. F. Fioriti, A. Hartmanns, and H. Hermanns. Partial order methods for statistical model checking and simulation. In *Formal Techniques for Distributed Systems*, pages 59–74. Springer, 2011.
5. B. Boyer, K. Corre, A. Legay, and S. Sedwards. PLASMA-lab: A flexible, distributable statistical model checking library. In K. Joshi, M. Siegle, M. Stoelinga, and P. D'Argenio, editors, *Quantitative Evaluation of Systems*, volume 8054 of *LNCS*, pages 160–164. Springer, 2013.
6. H. S. Chang, M. C. Fu, J. Hu, and S. I. Marcus. A survey of some simulation-based algorithms for Markov decision processes. *Communications in Information & Systems*, 7(1):59–92, 2007.
7. E. Clarke, E. A. Emerson, and J. Sifakis. Model checking: algorithmic verification and debugging. *Commun. ACM*, 52(11):74–84, Nov 2009.
8. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 3^{rd} edition, 2009.
9. H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal aspects of computing*, 6(5):512–535, 1994.
10. A. Hartmanns and M. Timmer. On-the-fly confluence detection for statistical model checking. In *NASA Formal Methods*, pages 337–351. Springer, 2013.
11. D. Henriques, J. G. Martins, P. Zuliani, A. Platzer, and E. M. Clarke. Statistical model checking for Markov decision processes. In 9^{th} *International Conference on Quantitative Evaluation of Systems (QEST2012)*, pages 84–93. IEEE, 2012.

12. W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.
13. W. G. Horner. A new method of solving numerical equations of all orders, by continuous approximation. *Philosophical Transactions of the Royal Society of London*, 109:308–335, 1819.
14. A. Israeli and M. Jalfon. Token management schemes and random walks yield self-stabilizating mutual exclusion. In *Proc. 9thAnnual ACM Symposium on Principles of Distributed Computing (PODC '90)*, pages 119–131. ACM New York, 1990.
15. C. Jegourel, A. Legay, and S. Sedwards. A platform for high performance statistical model checking – PLASMA. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 7214 of *LNCS*, pages 498–503. Springer, 2012.
16. C. Jegourel, A. Legay, and S. Sedwards. Importance splitting for statistical model checking rare properties. In N. Sharygina and H. Veith, editors, *Computer Aided Verification*, volume 8044 of *LNCS*, pages 576–591. Springer, 2013.
17. M. Kearns, Y. Mansour, and A. Y. Ng. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine Learning*, 49(2-3):193–208, 2002.
18. D. E. Knuth. *The Art of Computer Programming*. Addison-Wesley, 3^{rd} edition, 1998.
19. M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In M. Bernardo and J. Hillston, editors, *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*, volume 4486 of *LNCS (Tutorial Volume)*, pages 220–270. Springer, 2007.
20. M. Kwiatkowska, G. Norman, and D. Parker. Analysis of a gossip protocol in PRISM. *SIGMETRICS Perform. Eval. Rev.*, 36(3):17–22, Nov. 2008.
21. M. Kwiatkowska, G. Norman, D. Parker, and M. G. Vigliotti. Probabilistic mobile ambients. *Theoretical Computer Science*, 410(12-13):1272–1303, 2009.
22. R. Lassaigne and S. Peyronnet. Approximate planning and verification for large Markov decision processes. In *Proc.* 27^{th} *Annual ACM Symposium on Applied Computing*, pages 1314–1319. ACM, 2012.
23. A. Legay, S. Sedwards, and L.-M. Traonouez. Scalable verification of Markov decision processes. In 4^{th} *Workshop on Formal Methods in the Development of Software (FMDS 2014)*, LNCS. Springer, 2014.
24. U. Ndukwu and A. McIver. An expectation transformer approach to predicate abstraction and data independence for probabilistic programs. In *Proc.* 8^{th} *Workshop on Quantitative Aspects of Programming Languages (QAPL'10)*, 2010.
25. M. Okamoto. Some inequalities relating to the partial sum of binomial probabilities. *Annals of the Institute of Statistical Mathematics*, 10(1):29–35, 1958.
26. M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, 1994.
27. D. J. White. A survey of applications of Markov decision processes. *Journal of the Operational Research Society*, 44(11):1073–1096, Nov 1993.
28. H. L. S. Younes and R. G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *Computer Aided Verification*, pages 223–235. Springer, 2002.